The Accelerated Euclidean Algorithm

Sidi Mohamed Sedjelmaci

The Accelerated Euclidean Algorithm

Sidi Mohamed Sedjelmaci

Abstract

We propose a new GCD algorithm called Accelerated Euclidean Algorithm, or AEA for short, which matches the $O(n \log^2 n \log\log n)$ time complexity of Schönhage algorithm for n-bit inputs. This new GCD algorithm is designed for both integers and polynomials. We only focus our study to the integer case, the polynomial case is currently addressed ([3]).

1 Introduction

The algorithm is based on a half-gcd like procedure, but unlike Schönhage's algorithm, it is iterative and therefore avoids the penalizing calls of the repetitive recursive procedures. The new half-gcd procedure reduces the size the integers at least a half word-memory bits per iteration only in single precision. By a dynamic updating process, we obtain the same recurrence and the same time performance as in the Schönhage approach.

Throughout, the following notation is used. W is a word memory, i.e.: $W = 16$, 32 or 64. Let $u \geq v > 2$ be positive integers where u has a n bits with $n \geq 32$. Given a non-negative integer $x \in N$, $\ell(x)$ represents the number of its significant bits, not counting leading zeros, i.e.: $\ell(x) = \lceil \log_2(x+1) \rceil$. For the sake of readability integers U and V will be represented as a concatenation of l sets of W bits integers (except for the last set which may be shorter), with $l = \lceil n/W \rceil$, i.e.: if $U = \sum_{i=0}^{l-1} 2^{iW}\ U_{l-i}$ with $U_1 \neq 0$ and $V = \sum_{i=0}^{l-1} 2^{iW}\ V_{l-i}$, then we write symbollically $U = U_1 \bullet U_2 \ldots \bullet U_l$ and $V = V_1 \bullet V_2 \ldots \bullet V_l$.
We use specific vectors which represent interval subsets from of U and V, by:

$$X[i..j] = \begin{pmatrix} U_i \bullet U_{i+1} \bullet \cdots \bullet U_j \\ V_i \bullet V_{i+1} \bullet \cdots \bullet V_j \end{pmatrix} \ ; \ X[i] = \begin{pmatrix} U_i \\ V_i \end{pmatrix} \ ; \ \text{for } 1 \leq i < j \leq l.$$

Let $M(x)$ be the cost of a multiplication of two x-bit integers. The function $M(x)$ depends on the algorithm used to carry out the multiplications. The fast Schönhage-Strassen algorithm ([6]) performs these multiplications in $M(x) = O(n \log n \log\log n)$.

2 AEA: The Accelerated Euclidean Algorithm

The following algorithm is based on two main ideas:

- The computations are done in a Most Signifivant digit First (MSF) way.
- Update by multiplying, with the current matrix, `ONLY` twice the number of leading bits we have already chopped, `NOT` the others leading bits.

Algorithm `AEA`.

Input : $u \geq v > 2$ with $u \geq 8W$; $\quad n = \ell(u)$;
Output : a 2×2 matrix M and (U', V') such that
$M \times (U, V) = (U', V')$ and $\ell(V') \leq \ell(U) - 2^{\lfloor \log_2(n/W) \rfloor - 1} W$.
Begin

1. $(U, V) \leftarrow (u, v)$; $s \leftarrow \lfloor \log_2(n/W) \rfloor$;
2. **if** $\ell(V) \leq \lceil \ell(U)/2 \rceil + 1$ **return** I;
3. **if** $\ell(V) > \lceil \ell(U)/2 \rceil + 1$
4. **For** $i = 1$ **to** 2^{s-1}
5. **if** $U_i = 0$ *or* $V_i \neq 0$ (Regular case)
6. $h \leftarrow 0$;
7. **if** (i odd) $L_0 \leftarrow$ `ILE`$(X[i..i+1])$; **update**$_L(i, h)$;
8. **else**
9. $R_0 \leftarrow$ `ILE`$(X[i..i+1])$; **update**$_R(i, h)$;
10. $x \leftarrow i/2$; $h \leftarrow h + 1$;
11. **While** (x even)
12. $R_h \leftarrow R_{h-1} \times L_{h-1}$; **update**$_R(i, h)$;
13. $x \leftarrow x/2$; $h \leftarrow h + 1$;
14. **Enwhile**
15. $L_h \leftarrow R_{h-1} \times L_{h-1}$; **update**$_L(i, h)$;
16. **Endelse**
17. **else** `Irregular(i)` ($U_i \neq 0$ *and* $V_i = 0$) ;
18. **EndFor**
19. **Return** L_h and (U, V);

End

The algorithm `ILE` (borrowed from [7]) runs the extended Euclidean algorithm and stops when the remainder has roughly the half size of the inputs.
Algorithm `ILE`.

Input : $u_0 \geq u_1 \geq 0$
Output : a 2×2 matrix M and (u_{i-1}, u_i) such that $M \times (u_0, u_1) = (u_{i-1}, u_i)$ and $\ell(u_i) \leq \frac{1}{2}\ell(u_0)$.
Begin

1. $n = \ell(u_0)$; $\quad p = \ell(u_1)$;
2. **if** $p < \lceil n/2 \rceil + 1$ **return** $M = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$;
3. **if** $p \geq \lceil n/2 \rceil + 1$
4. $m = p - \lceil n/2 \rceil - 1$;
5. Apply Extended Euclid Algorithm until $|a_i| \leq 2^m < |a_{i+1}|$;
6. **return** $M = \begin{pmatrix} a_{i-1} & b_{i-1} \\ a_i & b_i \end{pmatrix}$ and (u_{i-1}, u_i).

End.

The functions $\mathbf{update}_L$ or $\mathbf{update}_R$ update not all the bits of the operands, but only the next useful small vectors, in order to get the next needed matrix. The irregular case is when $U_i \neq 0$ and $V_i = 0$, i.e.: one or many components are all equal to zero. Roughly speaking, we perform an euclidean division in order to make an efficient reduction and continue our process. The aim is to preserve, at most, the general scheme of our process. The basic idea is to use full updated vectors, i.e.: vectors updated with all the previous matrices L_h.

3 An Example

In order to carry out single-precision computations we take $W = 20$. Recall that the notation $\begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} a \bullet c \\ b \bullet d \end{pmatrix}$ means $\begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix} \times 2^W + \begin{pmatrix} c \\ d \end{pmatrix}$, where A, B, a, b, c and d are integers (c.f. notation in Section 1).

Let $\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 922375420941 \\ 707599307587 \end{pmatrix}$, then, with our notation, we obtain
$\begin{pmatrix} U \\ V \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 879645 \\ 674819 \end{pmatrix} \times 2^{20} + \begin{pmatrix} 785421 \\ 299843 \end{pmatrix} = \begin{pmatrix} 879645 \bullet 785421 \\ 674819 \bullet 299843 \end{pmatrix}$.

We have $U_1 = 879645$ and $V_1 = 674819$ then $\ell(U_1) = \ell(V_1) = 20$, $n = p = 20$ and $m_1 = p - 1 - \lceil \frac{n}{2} \rceil = 9$. Thus, in order to compute $ILE(U_1, V_1)$, we must stop the Extended Euclid Algorithm at $|a_i| \leq 2^9$. We obtain the matrix ($|a_i| < 2^9 = 512$)

$$N_1 = ILE(U_1, V_1) = \begin{pmatrix} 369 & -481 \\ -425 & 554 \end{pmatrix} \quad \text{then :}$$

$N_1 \begin{pmatrix} U \\ V \end{pmatrix} = N_1 \begin{pmatrix} U_1 2^{20} + U_2 \\ V_1 2^{20} + V_2 \end{pmatrix} = N_1 \begin{pmatrix} U_1 \\ V_1 \end{pmatrix} 2^{20} + N_1 \begin{pmatrix} U_2 \\ V_2 \end{pmatrix}$.

So $N_1 \times \begin{pmatrix} U_1 2^{20} + U_2 \\ V_1 2^{20} + V_2 \end{pmatrix} = \begin{pmatrix} 1066 \\ 601 \end{pmatrix} \times 2^{20} + \begin{pmatrix} 138 \\ -160 \end{pmatrix} \times 2^{20} + \begin{pmatrix} 892378 \\ 81257 \end{pmatrix}$

$$= \begin{pmatrix} 1204 \\ 441 \end{pmatrix} \times 2^{20} + \begin{pmatrix} 892378 \\ 81257 \end{pmatrix},$$

hence
$\begin{pmatrix} U_1 \\ V_1 \end{pmatrix} = \begin{pmatrix} 1204 \\ 441 \end{pmatrix}$ and $\begin{pmatrix} U_2 \\ V_2 \end{pmatrix} = \begin{pmatrix} 892378 \\ 81257 \end{pmatrix}$.

Now we can apply `ILE` to the new integers (less than 32 bits) $U = 1263377882$ and $V = 462503273$. We have $\ell(U) = 31$ and $\ell(V) = 29$. We obtain $m = 8$ and the second matrix (in 32-bits single precision)

$$N_2 = ILE(U, V) = \begin{pmatrix} -41 & 112 \\ 231 & -631 \end{pmatrix}$$

and $M = N_2 \times N_1 = \begin{pmatrix} -41 & 112 \\ 231 & -631 \end{pmatrix} \times \begin{pmatrix} 369 & -481 \\ -425 & 554 \end{pmatrix} = \begin{pmatrix} -62729 & 81769 \\ 353414 & -460685 \end{pmatrix}$.

Moreover, at this level, we may consider that U_1 and V_1 are eliminated (chopped), even if U_2 has some extra bits, and that U_2 and V_2 are updated as follows:

$$\begin{pmatrix} U_2 \\ V_2 \end{pmatrix} = \begin{pmatrix} 1873414 \\ 725479 \end{pmatrix} = \begin{pmatrix} 1 \bullet 824838 \\ 0 \bullet 725479 \end{pmatrix}; \quad \ell(U_2) = 21; \quad \ell(V_2) = 20.$$

On the other hand, it is easy to check that the final matrix $M = \begin{pmatrix} a_{t-1} & b_{t-1} \\ a_t & b_t \end{pmatrix}$ satisfies $M \begin{pmatrix} U \\ V \end{pmatrix} = \begin{pmatrix} -62729 & 81769 \\ 353414 & -460685 \end{pmatrix} \times \begin{pmatrix} 922375420941 \\ 707599307587 \end{pmatrix} = \begin{pmatrix} 1873414 \\ 725479 \end{pmatrix}$.

Now, if u and v were larger in size, namely:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 879645 \bullet 785421 \bullet u_3 \bullet u_4 \ \ldots \ \bullet u_k \\ 674819 \bullet 299843 \bullet v_3 \bullet v_4 \ \ldots \ \bullet v_k \end{pmatrix},$$

then we have to continue the half-gcd process. Since W bits have been already chopped, we have to update only the double, i.e.: multiply the next $2W$ leading bits of U and V by M, namely perform:

$\begin{pmatrix} U_3 \bullet U_4 \\ V_3 \bullet V_4 \end{pmatrix} \leftarrow M \times \begin{pmatrix} U_3 \bullet U_4 \\ V_3 \bullet V_4 \end{pmatrix}$, and disregard all the other bits of U and V.

Then do the same process as before with $\begin{pmatrix} U_2 \bullet U_3 \\ V_2 \bullet V_3 \end{pmatrix}$ instead of $\begin{pmatrix} U_1 \bullet U_2 \\ V_1 \bullet V_2 \end{pmatrix}$ to chop the vector $\begin{pmatrix} U_2 \\ V_2 \end{pmatrix}$, and so on, repeating the process till we reach the middle of the size of U.

4 An Example

Let us consider two consecutive Fibonacci numbers $(u, v) = (F_{59}, F_{58})$, i.e.:
$\begin{pmatrix} F_{59} \\ F_{58} \end{pmatrix} = \begin{pmatrix} 956722026041 \\ 591286729879 \end{pmatrix} = \begin{pmatrix} 956722 \\ 591286 \end{pmatrix} 10^9 + \begin{pmatrix} 026041 \\ 729879 \end{pmatrix} = \begin{pmatrix} 956722 \bullet 026041 \\ 591286 \bullet 729879 \end{pmatrix}$.
First, we must compute $mMAX$ which gives 2^{mMAX}, the maximum size of the output matrix L_1

$$mMAX = \ell(v) - \lceil \frac{\ell(u)}{2} \rceil - 1 = 19.$$

Let $(U_1, V1) = (956722, 591286)$ then $\ell(U_1) = \ell(V_1) = 20$, $n = p = 20$ and $m = p - 1 - \lceil \frac{n}{2} \rceil = 9$. We run the Extended Euclid algorithm and stops at $|a_i| \leq 2^9$. We obtain the matrix L_0 and the remainders (r_1, r_2):

$$L_0 = ILE(U_1, V_1) = \begin{pmatrix} -144 & 233 \\ 233 & -377 \end{pmatrix} \text{ and } (r_1, r_2) = (1670, 1404).$$

$$\text{Hence } L_0 \begin{pmatrix} U \\ V \end{pmatrix} = L_0 \begin{pmatrix} U_1 \ 10^9 + U_2 \\ V_1 \ 10^9 + V_2 \end{pmatrix} = L_0 \begin{pmatrix} U_1 \\ V_1 \end{pmatrix} 10^9 + L_0 \begin{pmatrix} U_2 \\ V_2 \end{pmatrix}.$$

By `update(1, 0)` we compute $L_0 \begin{pmatrix} U_2 \\ V_2 \end{pmatrix}$, so

$$L_0 \begin{pmatrix} U_1\ 10^9 + U_2 \\ V_1\ 10^9 + V_2 \end{pmatrix} = \begin{pmatrix} 1670 \\ 1404 \end{pmatrix} 10^9 + \begin{pmatrix} 166311903 \\ -269096830 \end{pmatrix} = \begin{pmatrix} 1836 \\ 1134 \end{pmatrix} 10^9 + \begin{pmatrix} 311903 \\ 903170 \end{pmatrix},$$

where `FIX` occurs in the last equality, hence the new values of U_1 and V_1:

$$\begin{pmatrix} U_1 \\ V_1 \end{pmatrix} \longleftarrow \begin{pmatrix} 1836 \\ 1134 \end{pmatrix} \text{ and } \begin{pmatrix} U_2 \\ V_2 \end{pmatrix} \longleftarrow \begin{pmatrix} 311903 \\ 903170 \end{pmatrix}.$$

We apply `ILE` to the new integers $U = 1836311$ and $V = 1134903$. We have $\ell(U) = 21$ and $\ell(V) = 21$. Repeating the same process as before, we obtain $m = 9$, the second matrix R_0 and remainders (r_1, r_2)

$$R_0 = ILE(U, V) = \begin{pmatrix} 233 & -377 \\ -377 & 610 \end{pmatrix} \text{ and } (r_1, r_2) = (2032, 1583).$$

Since $L_1 = R_0 \times L_0$ and using `update`(2,0) we obtain

$$L_1 \begin{pmatrix} U \\ V \end{pmatrix} = R_0 \begin{pmatrix} 1836311 \\ 1134903 \end{pmatrix} 10^6 + R_0 \begin{pmatrix} 903 \\ 170 \end{pmatrix} = \begin{pmatrix} 2032 \\ 1583 \end{pmatrix} 10^6 + \begin{pmatrix} 146309 \\ -236731 \end{pmatrix}.$$

$$\text{Thus } L_1 \begin{pmatrix} U \\ V \end{pmatrix} = \begin{pmatrix} 2178309 \\ 1346269 \end{pmatrix} = \begin{pmatrix} F_{32} \\ F_{31} \end{pmatrix}.$$

and $L_1 = R_0 \times L_0 = \begin{pmatrix} 233 & -377 \\ -377 & 610 \end{pmatrix} \times \begin{pmatrix} -144 & 233 \\ 233 & -377 \end{pmatrix} = \begin{pmatrix} -121393 & 196418 \\ 196418 & -317811 \end{pmatrix}$.

We can apply one more Euclid step because the matrix $Q \times L_1$ still satifies $|a_i| \leq 2^{mMAX} = 2^{19} = 524288$. So after one Euclid step we finally obtain:

$$Q \times L_1 \begin{pmatrix} U \\ V \end{pmatrix} = \begin{pmatrix} 1346269 \\ 832040 \end{pmatrix} = \begin{pmatrix} F_{31} \\ F_{30} \end{pmatrix},$$

and after $L_1 = Q \times L_1$, we have:

$$L_1 = \begin{pmatrix} 196418 & -317811 \\ -317811 & 514229 \end{pmatrix}.$$

Moreover, at this level, we may consider that U_1 and V_1 are eliminated (chopped), even if U_2 has some extra bits, and that U_2 and V_2 are updated as follows:

$$\begin{pmatrix} U_2 \\ V_2 \end{pmatrix} = \begin{pmatrix} 1873414 \\ 725479 \end{pmatrix} = \begin{pmatrix} 1 \bullet 824838 \\ 0 \bullet 725479 \end{pmatrix}; \quad \ell(U_2) = 21; \quad \ell(V_2) = 20.$$

On the other hand, it is easy to check that the final matrix $M = \begin{pmatrix} a_{t-1} & b_{t-1} \\ a_t & b_t \end{pmatrix}$ satisfies

$$M \begin{pmatrix} U \\ V \end{pmatrix} = \begin{pmatrix} -62729 & 81769 \\ 353414 & -460685 \end{pmatrix} \times \begin{pmatrix} 922375420941 \\ 707599307587 \end{pmatrix} = \begin{pmatrix} 1873414 \\ 725479 \end{pmatrix}.$$

Now, if u and v were larger in size, namely:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 879645 \bullet 785421 \bullet u_3 \bullet u_4 \ \ldots \ \bullet u_k \\ 674819 \bullet 299843 \bullet v_3 \bullet v_4 \ \ldots \ \bullet v_k \end{pmatrix},$$

then we have to continue the half-gcd process. Since W bits have been already chopped, we have to update only the double, i.e.: multiply the next $2W$ leading bits of U and V by M, namely perform:

$\begin{pmatrix} U_3 \bullet U_4 \\ V_3 \bullet V_4 \end{pmatrix} \leftarrow M \times \begin{pmatrix} U_3 \bullet U_4 \\ V_3 \bullet V_4 \end{pmatrix}$, and disregard all the other bits of U and V.

Then do the same process as before with $\begin{pmatrix} U_2 \bullet U_3 \\ V_2 \bullet V_3 \end{pmatrix}$ instead of $\begin{pmatrix} U_1 \bullet U_2 \\ V_1 \bullet V_2 \end{pmatrix}$ to chop the vector $\begin{pmatrix} U_2 \\ V_2 \end{pmatrix}$, and so on, repeating the process till we reach the middle of the size of U.

Here we stress that this is the main difference between our approach and the other Sorenson's like algorithms, where **all** the bits of U and V are updated by multiplying them with the matrix M. In our approach, we only update the double of what we have already chopped.

5 Remarks

Unlike the recursive versions of GCD algorithms (([1]),[4]), our aim is not to balance the computations on each leaf of the binary tree computations, but to make full of single precision everytime, computing therefore the maximum of quotients in single precision. This lead to different computations each step in the algorithm AEA and the other recursive GCD algorithms. Moreover the fundamental difference between AEA and Schönhage's approach is that AEA deals straightforward with the most significant leading bits first (**MSF computation**). Consequently, we can stop the algorithm AEA at any time and still obtain the leading bits of the result. Thus AEA is a strong MSF algorithm and can be considered for "on line" arithmetics, where all the basic operations can be carried out simultaneously as soon as only the needed bits are available. This new algorithm should be an alternative to the Schönhage GCD algorithm. On the other hand, the derived GCD algorithm deals with many applications where long euclidean divisions are needed. We have identified and started to study many applications such as subresultants and Cauchy index computation, *Pade*-approximates or LLL-algorithms.

6 References

[1] Gathen, J. von zur, Gerhard, G. Modern Computer Algebra. In *Cambridge University Press* (1999).

[2] D.H. Lehmer. Euclid's algorithm for large numbers, American Math. Monthly, 45, 1938, 227-233.

[3] M.F. Roy and S.M. Sedjelmaci. The Polynomial Accelerated Euclidean Algorithm, Subresultants and Cauchy index, work in progress, 2004.

[4] A. Schönhage. Schnelle Berechnung von Kettenbruchentwicklugen, Acta Informatica, 1, 1971, 139-144.

[5] A. Schönhage, A.,F.,W. Grotefeld and E. Vetter. Fast Algorithms, A Multitape Turing Machine Implementation, BI-Wissenschaftsverlag, Mannheim, Leipzig, Zürich, 1994.

[6] A. Schönhage, V. Strassen. Schnelle Multiplication grosser Zalen Computing 7,1971, 281-292.
[7] S.M., Sedjelmaci. On A Parallel Lehmer-Euclid GCD Algorithm, in Proc. of the International Symposium on Symbolic and Algebraic Computation (ISSAC'2001), 2001, 303-308.

Sidi Mohamed Sedjelmaci
LIPN CNRS UMR 7030,
Université Paris-Nord
Av. J.-B. Clément, 93430 Villetaneuse, France.
E-mail: *sms*@lipn.univ-paris13.fr